Amendments to the Specification:

Please replace the paragraph beginning at page 3, line 18 with the following amended paragraph:

The disclosed techniques are beneficial in the ~~contest~~ context of transport scenarios. Software is often developed on one system and distributed to other systems for use. One must guarantee that the correct version of test code is transported along with the correct version of production code. Here the test source code and production source code are embedded in the same code unit and are always transported together.

Please replace the paragraph beginning at page 6, line 14 with the following amended paragraph:

FIG. 2 shows an example of production source code 14 and test source code 16. The production source code 14 includes software instructions for implementing an application, such as a business program for an enterprise, whereas test source code 16 includes software instructions for testing the production source code 14 in the development environment 32. The example, which is written in ABAP programming language, includes an asterisk (*) for introducing comments in plain ~~test~~ text explaining the code and is disregarded by the compiler. The production source code 14 includes class definition 14a and class implementation 14b. The class definition 14a declares a production method ADD which is implemented in the respective class implementation 14b. In this example, the method ADD adds two import parameters A and B and yields the result in an export parameter RESULT.

Please replace the paragraph beginning at page 6, line 30 with the following amended paragraph:

The test class implementation 18b implements the test method TEST_ADD that is declared by the class definition. The test method TEST_ADD represents a method that tests the production method ADD of the production code 14. The test method TEST_ADD calls the production method ADD with parameter values, e.g., the parameter A set to a value of "3" and

the variable B set to a value of "5." The result of the call to the production method ADD is assigned to the variable ACTUAL_RESULT for subsequent comparison. The test class implementation 18b then uses the test assertion method ASSERT_EQUALS for comparing the value represented by ACTUAL_RESULT and ~~passes~~ passed to the parameter ACT with the expected value passed to the parameter EXP. This test assertion method is one of a series of assertions provided by utility class CL_AUNIT_ASSERT that is part of the ABAP programming environment. This utility class is part of the unit test framework "ABAP Unit" (Other test assertions are verified by methods from CL_AUNIT_ASSERT such as ASSERT_INITIAL, ASSERT_BOUND or just FAIL unconditionally). If the value passed to ACT is not equal to the value passed to EXP, then the test assertion method ASSERT_EQUALS generates a message with the ~~test~~ text specified by the parameter MSG indicating that the test failed or an error has been detected. If the value of ACT is equal to the value of EXP, then the test assertion method ASSERT_EQUALS returns without generating a message MSG, indicating that the test was successful. Thus, the production method ADD, if operating properly, returns a result of "8" based on adding "3" and "5." If the production method ADD is not operating properly, the test assertion method ASSERT_EQUALS detects this failure condition and yields a warning message.